Learning Analysis by Reduction from Positive Data*

František Mráz[1], Friedrich Otto[1], and Martin Plátek[2]

[1] Fachbereich Mathematik/Informatik, Universität Kassel
34109 Kassel, Germany
{mraz,otto}@theory.informatik.uni-kassel.de

[2] Charles University, Faculty of Mathematics and Physics
Department of Computer Science, Malostranské nám. 25
118 00 PRAHA 1, Czech Republic
Martin.Platek@mff.cuni.cz

Abstract. Analysis by reduction is a linguistically motivated method for checking correctness of a sentence. It can be modelled by restarting automata. In this paper we propose a method for learning restarting automata which are strictly locally testable (SLT-R-automata). The method is based on the concept of identification in the limit from positive examples only. Also we characterize the class of languages accepted by SLT-R-automata with respect to the Chomsky hierarchy.

1 Introduction

Analysis by reduction [7, 8, 11] consists in stepwise simplifications (reductions) of a given (lexically disambiguated) extended sentence until we get a correct simple sentence and accept or until an error is found and the input is rejected. Each simplification replaces a small part of the sentence by a shorter one. Analysis by reduction can be modelled by restarting automata, which have been studied for several years [6, 10]. On the other hand only few tools have been developed that support the design of restarting automata. In this paper we propose a method for learning restarting automata (analysis by reduction) from positive examples. Obviously, each restarting automaton learnt can also be used for language recognition. Moreover, analysis by reduction enables nice error localization in rejected words/sentences (see, e.g. [7]).

Several attempts for learning restarting automata by genetic algorithms were already made [2, 5]. The results are far from being applicable. Here we propose another method based on the concept of identification in the limit from positive data. Our proposed method uses positive samples of simplifications (reductions) and positive samples of so-called simple words (sentences) of the language to be learnt. The new algorithm could substantially improve applicability of restarting automata/analysis by reduction.

In this paper we describe the learning protocol for learning a subclass of restarting automata called strictly locally testable restarting automata. Their definition as well as their learning is based on the notion of strictly locally testable languages [12]. Further we compare the class of languages learnable in this way to the Chomsky hierarchy.

* The work of the first two authors was supported by a grant from the Deutsche Forschungsgemeinschaft. The third author was partially supported by the Grant Agency of the Czech Republic under Grant-No. 201/04/2102 and by the program 'Information Society' under project 1ET100300517.

2 Definitions and notations

In order to define the analysis by reduction, we introduce syntactic reduction systems.

Definition 1. A syntactic reduction system *is a tuple* $R = (\Sigma, \Gamma, \vdash_R, L_S)$*, where* Σ *is a finite nonempty* input alphabet*,* Γ *is a finite nonempty* working alphabet*,* $\Gamma \supseteq \Sigma$*,* $\vdash_R \subseteq \Gamma^* \times \Gamma^*$ *is the reduction relation, and* $L_S \subseteq \Gamma^*$ *is the set of* simple sentential forms*. Any string from* Γ^* *is called a* sentential form*. The reflexive and transitive closure of* $\vdash_R$ *is denoted by* $\vdash_R^*$*.*

For each syntactic reduction system $R = (\Sigma, \Gamma, \vdash_R, L_S)$ *we define two languages:*

- *the input language of* R: $L(R) = \{u \in \Sigma^* \mid \exists\ v \in L_S : u \vdash_R^* v\}$*, and*
- *the characteristic language of* R: $L_C(R) = \{u \in \Gamma^* \mid \exists\ v \in L_S : u \vdash_R^* v\}$.

That is, a word $u \in \Sigma^*$ (a sentential form $u \in \Gamma^*$) is in the input language (in the characteristic language) of R iff u can be reduced to some simple sentential form $v \in L_S$. Trivially, $L(R) = L_C(R) \cap \Sigma^*$.

Definition 2. *A syntactic reduction system* $R = (\Sigma, \Gamma, \vdash_R, L_S)$ *is called:*

- length-reducing *if, for each* $u, v \in \Gamma^*$*,* $u \vdash_R v$ *implies* $|u| > |v|$*;*
- locally-reducing *if there exists a constant* $k > 0$ *such that, for each* $u, v \in \Gamma^*$*,* $u \vdash_R v$ *implies that there exist words* $u_1, u_2, x, y \in \Gamma^*$ *for which* $u = u_1 x u_2$ *and* $v = u_1 y u_2$*, and* $|x| \leq k$*.*

Length- and locally-reducing syntactic reduction systems are a formalization of the analysis by reduction of the type we are interested in. In the case of a natural language, the relation $\vdash_R$ corresponds to a stepwise simplification of (extended) sentences, and L_S corresponds to (correct) simple sentences. The analysis by reduction is nondeterministic in the sense that:

- one word (sentential form) can be reduced in several different ways to different sentential forms;
- for a word $u \in L(R)$ there can exist two or more simple sentential forms to which u can be reduced;
- even if $u \in L_C(R)$, there can exist a sentential form v such that $u \vdash_R^* v$, but $v \notin L_C(R)$.

The analysis by reduction has the so-called *error preserving property*:

$$\text{if } u \vdash_R^* v, \text{ and } u \notin L_C(R), \text{ then } v \notin L_C(R).$$

Up-to now, the analysis by reduction has been modelled by several classes of restarting automata [10]. One of them is the RRWW-automaton ([7]). Instead of its formal definition we will use its alternative representation adapted from [9].

Definition 3. *A* restaring automaton *is a system* $M = (\Sigma, \Gamma, I)$*, where* Σ *is an input alphabet,* Γ *(*$\supseteq \Sigma$*) is a working alphabet, and* I *is a finite set of* meta-instructions *of the following two forms:*

(1) rewriting meta-instruction*:* $(E_l, x \to y, E_r)$*, where* $x, y \in \Gamma^*$ *such that* $|x| > |y|$ *and* $E_l, E_r \subseteq \Gamma^*$ *are regular languages called* left and right constraints*, or*
(2) accepting meta-instruction*:* (E, Accept)*, where* $E \subseteq \Gamma^*$ *is a regular language.*

Each restarting automaton $M = (\Sigma, \Gamma, I)$ *induces the following length- and locally-reducing syntactic reduction system* $R(M) = (\Sigma, \Gamma, \vdash_M^c, S(M))$*, where:*

a) for each $u, v \in \Gamma^*$*,* $u \vdash_M^c v$ *iff* $u = u_1 x u_2$*,* $v = u_1 y u_2$ *for some words* $u_1, u_2, x, y \in \Gamma^*$ *such that there exists an instruction* $i = (E_l, x \to y, E_r)$ *in* I *for which* $u_1 \in E_l$ *and* $u_2 \in E_r$*; and*
b) $S(M) = \bigcup_{(E,\mathsf{Accept})\in I} E$*.*

By $\vdash_M^{c^*}$ *we denote the reflexive and transitive closure of* $\vdash_M^c$*. A restarting automaton* $M = (\Sigma, \Gamma, I)$ *defines two languages:*

– *the input language of* M*:* $L(M) = \{w \in \Sigma^* \mid \exists z \in S(M) : w \vdash_M^{c^*} z\}$*, and*
– *the characteristic language of* M : $L_C(M) = \{w \in \Gamma^* \mid \exists z \in S(M) : w \vdash_M^{c^*} z\}$.

That is, an input word (a sentential form) w is accepted by M iff w can be reduced to some simple sentential form $z \in S(M)$.

The problem of learning analysis by reduction/a restarting automaton consists in learning the reduction relation $\vdash_M^c$ and the set of simple sentential forms $S(M)$. For simplicity, we will suppose a helpful teacher, which splits the problem of learning a restarting automaton into learning individual meta-instructions. Even knowing that $abababa \vdash_M^c ababa$ by some meta-instruction, we do not know whether a subword ab was replaced by the empty word λ, or a subword ba was replaced by λ, or some aba was rewritten to a, or some other rewriting was applied. Therefore, we suppose an even more helpful teacher which marks the rewritten part of such a word. In this way we reduce the problem of learning one meta-instruction to the learning of regular languages of constraints in meta-instructions. For this we can use one of the oldest models for language learning — the identification in the limit [4].

For a language L, a *positive presentation* of L is an infinite sequence $\{w_i\}_{i=1}^{\infty}$ of words from L such that every $w \in L$ occurs at least once in the sequence. Let $\mathcal{M}$ be a class of automata. Let $\mathcal{A}$ be an algorithm which, on input $\{w_1, \ldots, w_i\}$ (for any $i \geq 1$), returns a conjecture automaton $M_i \in \mathcal{M}$. An algorithm $\mathcal{A}$ is said to *learn (or identify) a language* L *in the limit from positive data using* $\mathcal{M}$ if for any positive presentation of L, the infinite sequence of automata $\{M_i\}_{i=1}^{\infty}$ in $\mathcal{M}$ produced by $\mathcal{A}$ satisfies the property that there exists an automaton M in $\mathcal{M}$ such that $L(M) = L$, and for all sufficiently large i, M_i is identical to M. A class of languages $\mathcal{L}$ is *learnable in the limit from positive data* (using $\mathcal{M}$) if there exists an algorithm $\mathcal{A}$ that, given an $L \in \mathcal{L}$, learns L in the limit from positive data using $\mathcal{M}$.

Gold [4] showed that any class of languages containing all finite sets and at least one infinite set is not learnable in the limit from positive data only. This fact implies that even the class of regular languages is not learnable in the limit from positive data. One of the well-known language classes which are learnable in the limit from positive data are the strictly locally testable languages [12].

In what follows, $|w|$ denotes the *length* of the word w, and $\subset$ denotes the proper subset relation. Reg, CFL, CSL denote the class of regular, context-free, and context-sensitive languages, respectively. Let $P_k(w)$ and $S_k(w)$ be the prefix and the suffix

of a word w of length k, resp. Further, let $I_k(w)$ be the set of all substrings of w of length k except the prefix and suffix of w of length k:

$$I_k(w) = \{u \mid |u| = k \text{ and } w = xuy, \text{ for some nonempty words } x, y\}.$$

These are defined only if $|w| \geq k$. If $|w| = k$, then $P_k(w) = S_k(w) = w$, while $I_k(w)$ is empty, if $k \leq |w| \leq k+1$. For example $P_2(aababab) = aa$, $S_2(aababab) = ab$, and $I_2(aababab) = \{ab, ba\}$.

Definition 4. *Let k be a positive integer. A language $L \subseteq \Sigma^*$ is strictly k-testable if there exist finite sets $A, B, C \subseteq \Sigma^k$ such that, for all $w \in L$ with $|w| \geq k$:*

$$w \in L \quad \textit{iff} \quad P_k(w) \in A,\ S_k(w) \in B,\ \textit{and } I_k(w) \subseteq C.$$

In this case, (A, B, C) is called a triple *for L.*

We will say that L is strictly locally testable *if it is strictly k-testable for some $k > 0$.*

Note, that the definition of "strictly k-testable" says nothing about the strings of length $k-1$ or less. Hence, L is strictly k-testable if and only if (see [12] for details)

$$L \cap \Sigma^k\Sigma^* = (A\Sigma^* \cap \Sigma^*B) - \Sigma^+(\Sigma^k - C)\Sigma^+. \tag{1}$$

For example, the language $(a+b)^*$ is strictly 1-testable, as $(a+b)^+$ can be expressed as (1) by $(A, B, C) = (\{a, b\}, \{a, b\}, \{a, b\})$, and the language $a(baa)^+$ is strictly 3-testable, as it can be expressed in the form (1) by the triple $(A, B, C) = (\{aba\}, \{baa\}, \{aba, baa, aab\})$.

We will denote the family of strictly k-testable languages by k-SLT and the class of strictly locally testable languages by SLT. It is easy to see that SLT $\subset$ Reg (E.g. the language $(aa)^*$ is not strictly locally testable). It is also known that

$$k\text{-SLT} \subset (k+1)\text{-SLT}.$$

Let us briefly recall a learning algorithm for strictly k-testable languages from [12]. First, we will present a construction which, for a given triple $S = (A, B, C)$, where $A, B, C \subseteq \Sigma^k$, constructs a deterministic finite state automaton (DFA) $M_S = (Q, \Sigma, \delta, q_0, F)$ such that $L(M_S) = (A\Sigma^* \cap \Sigma^*B) - \Sigma^+(\Sigma^k - C)\Sigma^+$:

$$Q := \begin{cases} Q_I \cup \{[\widehat{x}] \mid x \in A \cap B \cap (\Sigma^k - C)\} & \text{if } A \cap B \cap (\Sigma^k - C) \neq \emptyset, \\ Q_I & \text{otherwise} \end{cases}$$

where

$$\begin{aligned} Q_I := {} & \{[\lambda], [a_1], [a_1a_2], \ldots, [a_1 \ldots a_{k-1}] \mid a_1 \ldots a_k \in A, a_1, \ldots, a_k \in \Sigma\} \\ & \cup \{[x] \mid x \in A \cup B \cup C\}. \end{aligned}$$

Further,

$$q_0 := [\lambda],$$

$$F := \begin{cases} \{[\beta] \mid \beta \in B\} \cup \{[\widehat{x}] \mid x \in A \cap B \cap (\Sigma^k - C)\} & \text{if } A \cap B \cap (\Sigma^k - C) \neq \emptyset, \\ \{[\beta] \mid \beta \in B\} & \text{otherwise,} \end{cases}$$

and δ is defined as follows:

(i) for each $\alpha = a_1 \dots a_k \in A$, $(a_1, \dots, a_k \in \Sigma)$:

$$\delta(q_0, a_1) := [a_1],$$
$$\delta([w_i], a_{i+1}] := [w_i a_{i+1}] \qquad \text{where } w_i = a_1 \dots a_i (1 \le i \le k-2),$$
$$\delta([w_{k-1}], a_k) := \begin{cases} [\widehat{\alpha}] & \text{if } \alpha \in B \cap (\Sigma^k - C), \\ [\alpha] & \text{otherwise} \end{cases} \qquad \text{where } w_{k-1} = a_1 \dots a_{k-1};$$

(ii) for each $[ax], [xb] \in Q$ such that $|x| = k-1$, $ax \in A$, $xb \in B \cup C$

$$\delta([\widehat{ax}], b) := [xb] \qquad \text{if } ax \in B \cap (\Sigma^k - C),$$
$$\delta([ax], b) := [xb] \qquad \text{otherwise;}$$

(iii) for each $[ax], [xb] \in Q$ such that $|x| = k-1$, $ax \in C$, $xb \in B \cup C$

$$\delta([ax], b) := [xb].$$

We say that the constructed automaton M_S is *associated* to the triple $S = (A, B, C)$. The constructed DFA is in general not the minimal finite state automaton recognizing $L = L(M_S)$.

Now we present the learning algorithm LA (adapted from [12]).

Input: an integer $k > 0$ and a positive presentation of a target strictly k-testable language U.
Output: a sequence of DFAs accepting strictly k-testable languages.
Procedure:
initialize $E_0 := \emptyset$;
let $S_{E_0} = (\emptyset, \emptyset, \emptyset)$ be the initial triple;
construct DFA M_{E_0} accepting $E_0 (= \emptyset)$;
repeat (forever)
  let $M_{E_i} = (Q_{E_i}, \Sigma, \delta_{E_i}, q_0, F_{E_i})$ be the current DFA;
  read the next positive example w_{i+1};
  if $w_{i+1} \in L(M_{E_i})$ **then**
    $E_{i+1} := E_i$;
    $M_{E_{i+1}} := M_{E_i}$;
  else
    $E_{i+1} := E_i \cup \{w_{i+1}\}$;
    construct the DFA $M_{E_{i+1}}$ associated with the triple
      $S_{E_{i+1}} = (A_{E_{i+1}}, B_{E_{i+1}}, C_{E_{i+1}})$,
      where $A_{E_{i+1}} = A_{E_i} \cup P_k(w_{i+1})$,
        $B_{E_{i+1}} = B_{E_i} \cup S_k(w_{i+1})$,
        $C_{E_{i+1}} = C_{E_i} \cup I_k(w_{i+1})$.
  output M_{i+1}.

Yokomori and Kobayashi have shown that LA learns in the limit a DFA M_E such that $U = L(M_E)$.

Fact 5 ([12]) *Let $M_{E_0}, M_{E_1}, \ldots, M_{E_i}, \ldots$ be the sequence of* DFA*s produced by LA. Then*

1. *for each $i \geq 0$, $L(M_{E_i}) \subseteq M_{E_{i+1}} \subseteq U$, and*
2. *there exists $r > 0$ such that, for each $i \geq 0$, $M_{E_r} = M_{E_{r+i}}$, and $L(M_{E_r}) = U$.*

We now define a restricted version of restarting automata, which uses strictly k-testable languages only.

Definition 6. *Let k be a positive integer and Γ be an alphabet.*

- *We say that a rewriting meta-instruction $(E_l, x \to y, E_r)$, where $E_l, E_r \subseteq \Gamma^*$ and $x, y \in \Gamma^*$, is* strictly k-testable*, if the languages E_l, E_r are strictly k-testable and $k \geq |x| > |y|$.*
- *We say that an accepting meta-instruction (E, Accept), where $R \subseteq \Gamma^*$, is* strictly k-testable *if E is strictly k-testable.*
- *We say that a restarting automaton M is* strictly k-testable *if all its meta-instructions are strictly k-testable.*
- *We say that a restarting automaton is strictly locally testable, if it is strictly k-testable for some $k \geq 1$.*

Let k-SLT-R denote the class of all strictly k-testable restarting automata, and let SLT-R denote the class of all strictly locally testable restarting automata. For any class of restarting automata $\mathcal{A}$, $\mathcal{L}(\mathcal{A})$ denotes the class of all input languages recognized by automata from $\mathcal{A}$.

Let us give a sample SLT-R-automaton.

Example 1. The restarting automaton $M = (\{a, b\}, \{a, b\}, I)$, with the following three meta-instructions in I:

1. $(a + b, \mathsf{Accept})$
2. $(a^*, aa \to b, b^*)$
3. $(b^*, bb \to a, a^*)$

accepts the language

$$L(M) = \{a^{2i}b^j \mid i, j \geq 0, i + j = 2^n \text{ for some } n \geq 0\} \cup \{b^{2i}a^j \mid i, j \geq 0, i + j = 2^n \text{ for some } n \geq 0\}.$$

It is easy to check that all the constraints in the above instructions are strictly 2-testable languages. Hence, M is a strictly 2-testable restarting automaton.

So all SLT-R-automata can be learnt in the above proposed way. In the next section we will characterize the class of languages, which can be learnt in this way.

Note that according to [4] the class of strictly locally testable languages (SLT) as a whole is not learnable in the limit from only positive data. However, the inference algorithm can be effectively used to identify any language from this class in the limit through a complete (both positive and negative) presentation sequence of the language [3]. This can be accomplished by starting with $k = 2$ and using successive positive samples to infer progressively larger (less restricted) 2-SLT's until a

negative sample, which is incompatible with the current language, appears. Then k is increased by one, and the process continues in the same way with the successive samples. Eventually, the correct value of k will be reached and then no other negative sample will ever be incompatible. The inferred language will then grow progressively with the successive positive samples until the target k-SLT is identified.

The above learning protocol can be used also for learning strictly locally testable rewriting meta-instructions, but it requires a helpful teacher as follows. Suppose a learner is being taught a rewriting meta-instruction of the form $(E_l, aba \to b, E_r)$. Knowing that aba cannot be rewritten into b in the word $aabaaaa$, it is possible that:

- either, $a \notin E_l$, and $aaa \in E_r$ or
- $a \in E_l$, and $aaa \notin E_r$, or
- $a \notin E_l$, and $aaa \notin E_r$.

Hence, this information must be supplied by a teacher.

3 Characterization of the class of SLT-R-languages

SLT-R-automata are quite powerful. They can accept all growing context-sensitive languages (GCSL) as input languages. Growing context-sensitive languages are the languages generated by growing context-sensitive grammars. A Chomsky grammar $G = (V_N, V_T, S, P)$ with a set of nonterminals V_N, a set of terminals V_T, an initial nonterminal $S \in V_N$ and a set of rules P is a *growing context-sensitive grammar* if the start symbol S does not appear on the right-hand side of any production of G, and if $|\alpha| < |\beta|$ holds for all productions $(\alpha \to \beta) \in P$ satisfying $\alpha \neq S$.

Theorem 1. $\mathsf{GCSL} \subset \mathcal{L}(\mathsf{SLT\text{-}R}) \subseteq \mathsf{CSL}$.

Proof: Let $G = (V_N, V_T, S, P)$ be a growing context-sensitive grammar. Let us construct an SLT-R-automaton $M = (\Sigma, \Gamma, I)$ recognizing $L(M) = L(G)$. We take $\Sigma := V_T$, $\Gamma := (V_N \setminus \{S\}) \cup V_T$ and

$$I := \{(\{\beta \mid S \to \beta \in P\}, \mathsf{Accept})\} \cup \{(\Gamma^*, \beta \to \alpha, \Gamma^*) \mid \alpha \neq S, \text{ and } \alpha \to \beta \in P\}.$$

Trivially, M works as an analytical version of the grammar G without the rules with the initial nonterminal S, but M directly accepts (without any reduction) all right-hand sides of production with the left-hand side S. Hence $L(M) = L(G)$. This implies that $\mathsf{GCSL} \subseteq \mathcal{L}(\mathsf{SLT\text{-}R})$.

Moreover, the language $L_{copy} = \{w\#w \mid w \in \{a,b\}^*\}$ which is not even growing context-sensitive (see [1]), can be accepted by the following SLT-R-automaton $M_{copy} = (\Sigma, \Gamma, I_{copy})$, where $\Sigma = \{a, b, \#\}$, $\Gamma = \Sigma \cup \{A_{xy} \mid x, y \in \{a, b\}\}$ and I_{copy} consists of the following meta-instructions:

(1)	$(xy(a+b)^*\#, xy \to A_{xy}, (a+b)^*)$,	for each $x, y \in \{a, b\}$;
(2)	$(\lambda, xy \to A_{xy}, (a+b+\#A_{xy})^*)$,	for each $x, y \in \{a, b\}$;
(3)	$(A_{xy}(a+b)^*\#, A_{xy} \to \lambda, (a+b)^*)$,	for each $x, y \in \{a, b\}$;
(4)	$(\lambda, A_{xy} \to \lambda, (a+b+\#)^*)$,	for each $x, y \in \{a, b\}$;
(5)	$(\# + a\#a + b\#b, \mathsf{Accept})$.	

Each meta-instruction of M_{copy} preserves the number of occurrences of the symbol # in the current word. All words accepted by M_{copy} using the single accepting meta-instruction (5) contain exactly one occurrence of #. Hence, each word accepted by M_{copy} contains exactly one occurrence of #. For an input word w of the length at least 4 and of the form $(a+b)^*\#(a+b)^*$ the only applicable meta-instruction is a meta-instruction of the form (1). In the resulting word, # is followed by a working symbol A_{xy}, where $x, y \in \{a, b\}$ are the first two symbols of the word. Such a word can be reduced only by a meta-instruction of the form (2). Further, M_{copy} must use appropriate meta-instructions of the form (3) and (4). This is possible only if the word is of the form $w = xyu\#xyv$ for some $x, y \in \{a, b\}$, $u, v \in \{a, b\}^*$. After executing 4 meta-instructions, M_{copy} gets the word $u\#v$. Now it is easy to see, that M_{copy} accepts exactly the language L_{copy}. Moreover, all meta-instructions in I_{copy} are strictly 2-testable. Thus, $\mathsf{GCSL} \subset \mathcal{L}(\mathsf{SLT\text{-}R})$.

On the other hand, each restarting automaton can be simulated by a linear bounded automaton, from which it follows that $\mathcal{L}(\mathsf{SLT\text{-}R}) \subseteq \mathsf{CSL}$. □

For a restarting automaton $M = (\Sigma, \Gamma, I)$, the symbols from $\Gamma - \Sigma$ are called *auxiliary symbols*. From the practical point of view, the reduction analysis without auxiliary symbols enables the most transparent analysis with the best error localization.

Theorem 2. *Each regular language can be accepted by an* SLT-R*-automaton without auxiliary symbols.*

Proof: For each regular language $L \subseteq \Sigma^*$ there exists a finite state automaton A recognizing L. Let k denote the number of states of A. Reading a prefix of a word from Σ^* of length at least $k-1$ the automaton A must pass some state at least two times. Hence, for each word z of length $k-1$ there exist words $u, v, w \in \Sigma^*$ such that $z = uvw$, $v \neq \lambda$ and the automaton A is in the same state after reading both prefixes u and uv. Hence, for each word $x \in \Sigma^*$ it holds: $uvwx \in L$ iff $uwx \in L$. Then we can construct a $(k-1)$-SLT restarting automaton $M = (\Sigma, \Sigma, I)$ accepting L in the following way. M will have one accepting meta-instruction

$$(L \cap \Sigma^{<k-1}, \mathsf{Accept}),$$

which is $(k-1)$-SLT. For each $z \in \Sigma^{k-1}$, the automaton M will have a rewriting meta-instruction

$$(\{\lambda\}, z \to uw, \Sigma^*), \text{ where } z = uvw \text{ and } u, v, w \text{ are the words from above,}$$

which is also $(k-1)$-SLT.

It is easy to see that

- if $|x| < k-1$ then $x \in L$ iff $x \in L(M)$,
- if $|x| \geq k-1$ then there exists a word $x' \in \Sigma^*$, $|x'| < |x|$ such that $x \vdash_M^c x'$ and $(x \in L$ iff $x' \in L)$.

From this it follows, that $L(M) = L$.

□

Note, that the SLT-R-automaton M constructed in the above proof uses only a finite language in its single accepting meta-instruction and all its rewriting instruction enable only the language $\{\lambda\}$ in their left constraints.

The above theorem shows that $\mathsf{Reg} \subseteq \mathcal{L}(\mathsf{SLT\text{-}R})$. This inclusion is proper even if we consider SLT-R-automata without auxiliary symbols (or recognizing characteristic languages). Such automata can accept even some non-context-free languages (see the automaton M from Example 1). Theorem 1 implies that each context-free language can be accepted by an SLT-R-automaton. Unfortunately, restarting automata without auxiliary symbols cannot accept all context-free languages. In [7] it is shown that the context-free language

$$L = \{a^n b^n \mid n \geq 0\} \cup \{a^n b^m \mid m > 2n \geq 0\}$$

cannot be accepted by any restarting automaton without auxiliary symbols. Hence we get the following theorem.

Theorem 3. $\mathsf{CFL} \subset \mathcal{L}(\mathsf{SLT\text{-}R})$*, but there exist context-free languages which cannot be accepted by any* SLT-R*-automaton without auxiliary symbols.*

4 Conclusions

There are many possible approaches for learning analysis by reduction/restarting automata. Our proposed method which reduces it to learning the corresponding set of meta-instructions has several advantages:

1. The whole task of learning a restarting automaton can be split into smaller tasks of learning one meta-instruction at a time. Learning several simpler meta-instructions should be computationally simpler than learning the whole language at once.
2. For learning different meta-instructions we can use different models and algorithms for learning regular languages.
3. The learning can be done in an incremental way. First we can learn some basic meta-instructions which define only a subset of the target language. Then we can continue to learn new meta-instructions to improve our approximation of the target language.
4. A rewriting meta-instruction of a restarting automaton M is called *correctness preserving* if, for each rewriting $u \vdash_M^c v$ according to this meta-instruction, it holds that $u \in L_C(M)$ iff $v \in L_C(M)$. If we succeed to learn correctness preserving meta-instructions, then it is possible to learn the target language in parallel. That is, two or more (correctness preserving) meta-instructions can be learned separately and finally put together in one automaton.

The proposed approach can use any known algorithm for learning regular languages. Accordingly, we plan to also consider other learning protocols like learning from positive and negative examples, learning using membership and equivalence queries, etc.

References

1. G. Buntrock and F. Otto. Growing context-sensitive languages and Church-Rosser languages. *Information and Computation*, 141(1):1–36, 1998.
2. J. Čejka. Learning correctness preserving reduction analysis. BSc. project, Faculty of mathematics and physics, Charles university, 2003. In Czech.
3. P. Garcia and E. Vidal. Inference of k-testable languages in the strict sense and application to syntactic pattern recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(9):920–925, 1990.
4. E. M. Gold. Language identification in the limit. *Inf. and Control*, 10(1967), 447–474.
5. P. Hoffmann. Learning restarting automata by genetic algorithms. In M. Bieliková, ed., *SOFSEM 2002: Student research forum*, Milovy, Czech Republic, 2002, 15–20.
6. P. Jančar, F. Mráz, M. Plátek, and J. Vogel. Restarting automata. In H. Reichel, ed., *FCT'95, Proc.*, LNCS **965**, Springer 1995, 283–292.
7. P. Jančar, F. Mráz, M. Plátek, and J. Vogel. On monotonic automata with a restart operation. *Journal of Automata, Languages and Combinatorics*, 4(1999), 287–311.
8. M. Lopatková, M. Plátek, and V. Kuboň. Modeling syntax of free word-order languages: Dependency analysis by reduction. In V. Matoušek, P. Mautner , and T. Pavelka, eds, *TSD 2005, Proc.*, LNCS **3658**, Springer, 2005, 140–147.
9. G. Niemann and F. Otto. On the power of RRWW-automata. In M. Ito, G. Păun, and S. Yu, eds., *Words, Semigroups, and Transductions*. World Scientific, Singapore, 2001, 341–355.
10. F. Otto. Restarting automata and their relation to the Chomsky hierarchy. In Z. Ésik and Z. Fülöp, eds., *DLT 2003, Proc.*, LNCS **2710**, Springer, 2003, 55–74.
11. M. Plátek, M. Lopatková, and K. Oliva. Restarting automata: motivations and applications. In: M. Holzer (ed.), *Workshop 'Petrinetze' and 13. Theorietag 'Formale Sprachen und Automaten', Proc.*, Institut für Informatik, Technische Universität München, 2003, 90–96.
12. T. Yokomori, and S. Kobayashi. Learning local languages and their application to DNA sequence analysis. IEEE Trans. on Pattern Anal. and Machine Intell., 20(1998), 1067–1079.